

METHOD, SYSTEM AND COMPUTER PROGRAM PRODUCT FOR
STORING transliteration and/or phonetic spelling
information in a text string class

RELATED APPLICATIONS

5

INS. A1
10 The present invention is related to the subject matter
of the following commonly assigned, copending United States
patent applications: serial no. 08/_____ (Docket No. AT9-
98-108) entitled "METHOD, SYSTEM, AND COMPUTER PROGRAM
PRODUCT FOR PROVIDING A USER INTERFACE WITH ALTERNATIVE
15 DISPLAY LANGUAGE CHOICES" and filed _____, 1998; serial
no. 08/_____ (Docket No. AT9-98-158) entitled "METHOD,
SYSTEM AND COMPUTER PROGRAM PRODUCT FOR CAPTURING LANGUAGE
TRANSLATION AND SORTING INFORMATION INTO A TEXT STRING
CLASS" and filed _____, 1998; serial no. 08/_____
20 (Docket No. AT9-98-159) entitled "METHOD, SYSTEM AND
COMPUTER PROGRAM PRODUCT FOR SORTING TEXT STRINGS" and filed
_____, 1998; serial no. 08/_____ (Docket No. AT9-98-
408) entitled "METHOD, SYSTEM AND COMPUTER PROGRAM PRODUCT
FOR ROTATING THROUGH A SEQUENCE OF DISPLAY STATES IN A
MULTI-FIELD TEXT CLASS IN A GRAPHICAL USER INTERFACE" and
25 filed _____, 1998; serial no. 08/_____ (Docket No. AT9-
98-409) entitled "METHOD, SYSTEM AND COMPUTER PROGRAM
PRODUCT FOR CONTROLLING THE GRAPHICAL DISPLAY OF MULTI-FIELD
TEXT STRING OBJECTS" and filed _____, 1998; serial no.
08/_____ (Docket No. AT9-98-410) entitled "METHOD, SYSTEM
AND COMPUTER PROGRAM PRODUCT FOR DISPLAYING THE CONTENTS OF
30 ALL FIELDS IN A MULTI-FIELD TEXT STRING OBJECT" and filed
_____, 1998; serial no. 08/_____ (Docket No. AT9-98-
411) entitled "METHOD, SYSTEM AND COMPUTER PROGRAM PRODUCT
FOR DYNAMIC LANGUAGE SWITCHING IN A MULTI-FIELD TEXT STRING

OBJECT VIA MESSAGING" and filed _____, 1998; and serial
no. 08/_____ (Docket No. AT9-98-578) entitled "METHOD,
SYSTEM AND COMPUTER PROGRAM PRODUCT FOR AUTOMATIC CHARACTER
TRANSLITERATION IN A TEXT STRING OBJECT" and filed _____,
1998. The content of the above-referenced applications is
incorporated herein by reference.

BACKGROUND OF THE INVENTION

1. Technical Field:

The present invention relates in general to text strings in data processing systems and in particular to encapsulation of identification, meaning or pronunciation information in text strings. Still more particularly, the present invention relates to a multi-field text string encapsulating identification, meaning, and pronunciation information utilizing different character sets.

2. Description of the Related Art:

Multinational companies often run information system (IS) networks which span multiple countries spread around the globe. To maximize the usefulness of such networks, operations within each locale tend to run in the local language of the region. Where possible, names of abstract objects in user applications are in the local language and match the local language, organization, city, or human names which the abstract objects represent. In the case of system management software, often abstract objects would represent each of a global enterprise's local offices.

Central management of such a global network may be difficult or impossible when abstract object names utilize the local language and the local language's underlying character set. For offices located in Egypt, abstract objects would most naturally be named in Arabic; offices in Russia would name objects utilizing the Cyrillic character set; and for offices in Japan, objects would be named in Japanese. A problem arises, however, when a enterprise's headquarters IS staff attempts to examine these objects. The IS staff at the multinational headquarters located in the United States is unlikely to be able to read Arabic or Japanese, or even recognize Cyrillic characters.

Japanese, for example, is a logosyllabic or ideographic language which does not have an alphabet representing simple sounds, but instead has a very large character set with symbols ("ideographs") corresponding to concepts and objects rather than simple sounds. For instance, the Joyo Kanji List (Kanji for Daily Use) adopted for the Japanese language in 1981 includes 1945 symbols. Users unfamiliar with the Kanji characters will have difficulty identifying a particular abstract object named in Japanese, as well as difficulty even discussing such abstract objects over the telephone with an English- and Japanese-speaking counterpart.

Additionally, merely seeing an ideograph may provide no clue as to the correct meaning or pronunciation since, in Japanese, the same character may have multiple meanings or pronunciations. For instance, the character depicted in **Figure 7A** may mean either "West" or "Spain"; the symbol depicted in **Figure 7B** may be pronounced either "hayashi" or

"rin" (or "lin"); and the characters depicted in **Figure 7C** may be pronounced "suga no," "suga ya," "kan no," or "kan ya." This circumstance is based in part on the history of the Japanese language, in which the Kanji characters were adopted from the Chinese language in several waves. Thus, for example, the "rin" symbol depicted in **Figure 7B** is On-Yomi, basically a simulation of the Chinese pronunciation when the character was imported to Japan, while "hayashi" is Kun-Yomi, a Japanese word assigned to the character which has the same meaning.

It would be desirable, therefore, to provide a data processing system text string encapsulating identification, meaning, and pronunciation information utilizing different character sets.

SUMMARY OF THE INVENTION

5 It is therefore one object of the present invention to provide improved method, system and computer program product for storing text strings in data processing systems.

10 It is another object of the present invention to provide a method, system and computer program product for encapsulation of identification, meaning or pronunciation information in text strings employed by data processing systems.

15 It is yet another object of the present invention to provide a method, system and computer program product for implementing a multi-field text string encapsulating identification, meaning, and pronunciation information utilizing different character sets.

20 The foregoing objects are achieved as is now described. A multi-field text string data structure is employed to encapsulating identification, meaning, and pronunciation and/or sorting information for a text string. A first field contains the characters for the text string in a language in which the text string is entered, which may be latin characters, characters which sound-map to latin characters, or one or more ideographs. A second field contains either the same characters or an intermediate representation of the text string, such as syllabary characters for a phonetic spelling of the characters within the first field. A third field contains either the same characters as the first field or a latin character phonetic spelling of the characters in the first field. The first field thus contains the text

5

10

[illegible]

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself however, as well as a preferred mode of use, further objects and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

Figure 1 depicts a diagram of a data processing system in which a preferred embodiment of the present invention may be implemented;

Figure 2 is a diagram of a multi-field text string class employed in encapsulating transliteration and/or phonetic spelling information in accordance with a preferred embodiment of the present invention;

Figure 3 depicts a high level flowchart for a process of entering data into a multi-field text string class in accordance with a preferred embodiment of the present invention;

Figures 4A-4B are portions of a user interface showing one application for character-mapped data entry into alternate fields of a multi-field text string class in accordance with a preferred embodiment of the present invention;

Figure 5 is a high level flowchart for a process of data entry in a logossyllabic language into a multi-field

text string class in accordance with a preferred embodiment of the present invention;

5 **Figure 6A-6G** are illustrations of user interface displays for a process of data entry in a logossyllabic language into a multi-field text string class in accordance with a preferred embodiment of the present invention; and

10 **Figure 7A-7C** are pictorial representations of known Japanese ideographs which may have multiple meanings or pronunciations.

SECRET

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

5 With reference now to the figures, and in particular
with reference to **Figure 1**, a block diagram of a data
processing system in which a preferred embodiment of the
present invention may be implemented is depicted. Data
processing system 100 may be, for example, one of the
Aptiva® models of personal computers available from
International Business Machines Corporation of Armonk, New
York. Data processing system 100 includes a processor 102,
10 which in the exemplary embodiment is connected to a level
two (L2) cache 104, which is connected in turn to a system
bus 106. In the exemplary embodiment, data processing
system 100 includes graphics adapter 118 connected to system
bus 106, receiving user interface information for display
120.

Also connected to system bus 106 is system memory 108
and input/output (I/O) bus bridge 110. I/O bus bridge 110
couples I/O bus 112 to system bus 106, relaying and/or
transforming data transactions from one bus to the other.
Peripheral devices such as nonvolatile storage 114, which
may be a hard disk drive, and keyboard/pointing device 116,
which may include a conventional mouse, a trackball, or the
like, are connected to I/O bus 112.

25 The exemplary embodiment shown in **Figure 1** is provided
solely for the purposes of explaining the invention and
those skilled in the art will recognize that numerous
variations are possible, both in form and function. For
instance, data processing system 100 might also include a
compact disk read-only memory (CD-ROM) or digital video disk

(DVD) drive, a sound card and audio speakers, and numerous other optional components. All such variations are believed to be within the spirit and scope of the present invention. Data processing system 100 and the Java implementation examples below are provided solely as examples for the purposes of explanation and are not intended to imply architectural limitations. Those skilled in the art will recognize the numerous programming languages which may be utilized, all of which are believed to be embraced within the spirit and scope of the invention.

Referring to **Figure 2**, a diagram of a multi-field text string class employed in encapsulating transliteration and/or phonetic spelling information in accordance with a preferred embodiment of the present invention is depicted. A fundamental problem in multinational computing environments which need to display data in multiple human languages is that a spoken word generally encapsulates information in multiple aspects or attributes, such as through the word's meaning, from context, and/or from inflection. When reduced to a visual or electronic representation for manipulation or display in a data processing system, the word may lose some attributes and much of the associated meaning. Most importantly for data processing systems, a visual representation of a word may give no clues as to the correct translation or pronunciation of the word or the proper placement of a word within a specified sort order. International String ("IString") class 202 may be employed to address this problem.

IString class 202 is preferably a Java class similar to the Java String class, which behaves like the String class

by including similar methods. Most of the original behavior of the String class should be preserved, with additional functionality added and utilized only as needed. IString class 202 is a datatype which captures some of the meaning of spoken words which is normally lost when the word is reduced to a visual representation. IString class 202 is preferably utilized for all object names and system messages within a system.

The IString class 202 structure includes three different strings for each name, message, data, or text object: a baseString 204, a sortString 206, and an altString 208. BaseString 204 is the string within IString class 202 employed by default in the user interface display and may contain any text, usually the original text entered by the user in the local language where the IString object is created. SortString 206 may also be any text and is employed to allow correct sorting of non-phonetic languages and languages which are difficult to sort based only on the binary value of baseString 204. AltString 208 may be any text but should conventionally be filled with a latin character set representation of the pronunciation of the data contained in baseString 204. Thus, IString class 202 includes the original text (baseString 204), a sort key (sortString 206), and a pronunciation key (altString 208) for object names, system messages, and other data.

When implemented in Java, a constructor for an IString class 202 object may be composed of the following fields:

/** The base text String */
protected String baseString;

/** The related text String for proper collation */
protected String sortString;

/** The related alternate text String (pronunciation key) */
protected String altString;

/** The source locale, as an ISO-3166 code; used for collation */
protected String sourceLocale;

/** The source language, as an ISO-639 code */
protected String sourceLanguage;

/** The source variant defined for EBCDIC and case mapping */
protected String sourceVariant;

/** The target locale, as an ISO-3166 code */
protected String targetLocale;

/** The target language, as an ISO-639 code */
protected String targetLanguage;

/** The target variant defined for EBCDIC and case mapping */
protected String targetVariant;

Complete listings of the upper-case, two letter ISO Country Codes defined by ISO-3166 and the lower-case, two letter ISO Language Codes defined by ISO-639 are readily available from a variety of sources on the Internet.

Table I illustrates how data within the IString data type 202 looks when represented as a table:

| Field | Type | Data |
|----------------|-------------|-----------------------------|
| baseString | Java String | The user's text |
| sortString | Java String | Language/locale dependent |
| altString | Java String | Language/locale dependent |
| sourceLocale | Java String | ISO-3166 code, example "US" |
| sourceLanguage | Java String | ISO-639 code, example "en" |
| sourceVariant | Java String | Variant code |
| targetLocale | Java String | ISO-3166 code, example "JP" |
| targetLanguage | Java String | ISO-639 code, example "ja" |
| targetVariant | Java String | Variant code |

Table I

A Java constructor for a new, empty IString class object 202 where the contents are independent of language or locale may be:

```

/*****
 *
 * <P></P>
 *
 * <dt><b>Description:</b><dd>
 * <p> Allocate a new IString containing no characters in the default
 * locale.</p>
 *
 *****/
public IString() {
    this.baseString = new String();
    this.sortString = new String();
    this.altString = new String();
    init();
}

```

To allow objects of the IString class 202 datatype to be stored in an Object Database (ODB), however, and to permit manipulation of IString data by Common Object Request Broker

Architecture (CORBA) applications, an Interface Definition Language (IDL) class should be defined:

```

    struct IString{
    string baseString;           //base test String
    string sortString;          //related text String for collation
    string altString;           //related alternate text String (pronunciation)
    string sourceLocale;        //source locale as an ISO-3166 code
    string sourceLanguage;      //source language as an ISO-639 code
    string sourceVariant;       //source variant code
    string targetLocale;        //target locale as an ISO-3166 code
    string targetLanguage;      //target language as an ISO-639 code
    string targetVariant;       //target variant code
    }

```

The contents of baseString 204, sortString 206, and altString 208 are preferably but not necessarily Unicode text entered by data entry methods 210 within IString class 202. Data entry methods 210, and thus the contents of baseString 204, sortString 206, and altString 208, may depend at least in part on language and locale parameters defined by sourceLocale field 212, sourceLanguage field 214, targetLocale field 216, and targetLanguage 218.

Because data entry methods 210 are dependent on the locale and/or language employed by the underlying host system, creation of a new IString object 202 preferably results in the locale and language properties of the host system in which the IString object 202 is created being placed in sourceLocale field 212 and sourceLanguage field 214. A constructor for allocating a new, empty IString for a specified locale and language determined from the host system in which the IString class object 202 is being created may be:

```

/*****
*
* <P></P>
*
* <dt><b>Description:</b><dd>
5 * <p> Allocate a new IString containing no characters in the
* specified locale.</p>
*
*****
10 public IString(Locale loc) {
    this.baseString = new String();
    this.sortString = new String();
    this.altString = new String();
    this.sourceLocale = loc.getLocale();
    this.sourceLanguage = loc.getLanguage();
    init();
    }
15
20
25
30

```

Input of data into an IString class 202 object is preferably locale- or language-dependent. The source-Language and targetLanguage properties 214 and 218 control how data is input into an IString class object 202 by data input methods 210. The sourceLanguage property 214 may be set to the language property of the host system on which the IString class object is created. The targetLanguage property 218 may also be set to that language, or may alternatively be set to a common, "universal" language such as English. Data input methods 210 compare sourceLanguage and targetLanguage properties 214 and 218 to determine what is entered into baseString 204, sortString 206, and altString 208 in an IString class object 202.

Character strings are entered into the baseString 204, sortString 206, and altString 208 fields by data input methods 220 for IString class 202, which may selectively utilize data from either the user's direct entry or

specification, from transliteration engine 220, or from the Input Method Editor (IME) 224. Where the targetLanguage property 218 is set to English as a default, data entry methods 210 determine the contents of baseString 204, sortString 206, and altString 208 fields based upon the character set employed by the language in which data is entered by the user (sourceLanguage property 214).

For languages which employ the latin character set, the user input is placed by data entry methods 220 into all three fields (baseString 204, sortString 206, and altString 208) of the IString class 202 by data entry methods 210. A suitable constructor may be:

```

/*****
 *
 * <P></P>
 *
 * <dt><b>Description:</b><dd>
 * <p> Allocate a new IString which contains the same sequence of
 * characters as the string argument in the specified locale.</p>
 *
 *****/
public IString(String str, Locale loc) {
    this.baseString = new String(str);
    this.sortString = new String(str);
    this.altString = new String(str);
    this.sourceLocale = loc.getLocale();
    this.sourceLanguage = loc.getLanguage();
    init();
}

```

For most locales and languages, the entered string will be input into all three fields of the IString object 202. If targetLanguage property 218 were not set to English, data entry methods 224 would input the user-entered text into all three fields whenever the languages identified in sourceLanguage and targetLanguage properties 214 and 218 employ a

common character set (e.g., both employ latin characters, as in the case of Spanish and Afrikaans).

5 **Table II** illustrates how data is entered into IString class 202 fields where the host language and locale utilize the latin character set.

| Field | Type | Data |
|----------------|-------------|--------------|
| baseString | Java String | Hetherington |
| sortString | Java String | Hetherington |
| altString | Java String | Hetherington |
| sourceLocale | Java String | US |
| sourceLanguage | Java String | en |
| targetLocale | Java String | US |
| targetLanguage | Java String | en |

10
15
20
25
30
Table II

If desired, the fields may be individually edited and the object artificially promoted for sorting purposes by inserting a string having a lower sort value (e.g., "AAA_Hetherington") into sortString 206.

For languages which do not employ the latin character set, but which utilize a character set which may be sound mapped to the latin character set, the user input is entered by data entry methods 210 into baseString 204 and sortString 206, but a transliterated, phonetic representation of the input is placed in altString 208. An internal method within the transliteration engine 220 is employed to sound-map the passed string to a phonetic, latin character representation for altString 208 to transliterate entered characters into

other characters understandable to people who are not familiar with the character set of the original language.

5 To generate the contents of altString 208, transliteration engine 220 selects an appropriate Java resource file 222 containing a mapping table to create the alternate text to be placed in altString 208. The selection of the particular resource file which is employed based on the combination of source and target languages. Java resource files 222 are named for the combination of languages for which the mapping is being performed. In the example shown in Figure 2, ru_en.class is for mapping Russian (Cyrillic characters) to English (Latin characters). The structure of resource file 222 is a table with associated entries for foreign language characters and corresponding latin characters.

10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
A suitable constructor for an IString object in which altString 208 is transliterated from the passed string may be:

```

/*****
*
* <P> </P>
*
* <dt> <b>Description:</b> <dd>
* <p> Allocate a new IString. The baseString and sortString are the
* passed string, the altString is transliterated into the target
* language.</p>
*
*****/
public IString(String str) {
    this.baseString = new String(str);
    this.sortString = new String(str);
    if(isSameLanguage())
        this.altString = new String(str);
    else
        this.altString = transmogrify(str,
                                     this.sourceLanguage,
                                     this.targetLanguage);
}

```

The "transmogrify" method is the internal method within transliteration engine 220 which was described above. The character set into which the entered characters are transliterated is determined from the targetLanguage property 218, which in the exemplary embodiment is assumed to be set to English. Given an appropriate resource file 222, however, characters may be transliterated between any two languages for which characters in one language sound-map to one or more characters in the other.

Table III illustrates how data is entered into IString class 202 by data entry methods 210 where the language utilizes a non-latin character set which maps to the latin character set, such as Russian Cyrillic.

| Field | Type | Data |
|----------------|-------------|--------------|
| baseString | Java String | Давид Кумгир |
| sortString | Java String | Давид Кумгир |
| altString | Java String | David Kumhyr |
| sourceLocale | Java String | RU |
| sourceLanguage | Java String | ru |
| targetLocale | Java String | US |
| targetLanguage | Java String | en |

Table III

In the example shown, the text entered by the user is inserted into both baseString 204 and sortString 206, but the text entered into altString 208 is selected by transliteration engine 220 utilizing a resource table of Russian Cyrillic to English character sound mappings. The phonetic representation of the baseString 204 is thus entered into altString 208 as a pronunciation key for users unfamiliar with the Cyrillic character set.

For languages which do not employ the latin character set or a character set which may be sound-mapped to the latin character set, data entry methods 210 input data into the baseString 204, sortString 206, and altString 208 fields which is derived from the input method editor (IME) 224. IME 224 may be either a customized input method editor or the input method editor which is integrated into Asian versions of the Windows NT operating system available from Microsoft Corporation of Redmond Washington. If the Windows NT input method editor is employed, the appropriate data must be extracted from the Windows NT input method editor internal data storage.

Table IV illustrates how data is entered into IString class 202 by data entry methods 210 for logosyllabic languages, such as Japanese, which employ neither the latin character set nor a character set which may be sound-mapped to the latin character set.

| Field | Type | Data |
|----------------|-------------|---------|
| baseString | Java String | <Kanji> |
| sortString | Java String | はやし |
| altString | Java String | hayashi |
| sourceLocale | Java String | JP |
| sourceLanguage | Java String | ja |
| targetLocale | Java String | US |
| targetLanguage | Java String | en |

Table IV

Logosyllabic languages do not have alphabets, but instead have very large character sets with symbols ("ideographs") corresponding to concepts and objects rather than simple sounds. For instance, the Joyo Kanji List (Kanji for Daily Use) adopted for the Japanese language in 1981 includes 1945 symbols. Normal computer keyboards cannot contain enough separate keys to have one for each symbol in the language, so input is accomplished phonetically utilizing keystroke combinations to select characters from one of two phonetic syllabaries, hiragana or katakana, and dictionary lookup for Kanji symbol creation. The process is implemented in the Windows NT input method editor identified above.

For logosyllabic or ideographic languages, therefore, the data entered into altString 208 is the latin characters typed by the user to compose the desired ideograph. The

data entered into sortString 206 are the syllabary characters phonetically spelling the desired ideograph, providing an intermediate representation of the ideograph. The data entered into baseString 204 is the final ideograph selected by the user. As with transliteration of non-latin characters as described above, non-latin characters may be entered into altString 208 if the targetLanguage property is set to a language other than English and IME 224 supports composition of the ideographs by phonetic spelling in a language other than English. For instance, an IString object 202 might contain Japanese Kanji in baseString 204, hiragana in sortString 206, and Cyrillic characters in altString 208 if IME 224 permits composition of Japanese Kanji characters by phonetic spelling in Russian.

A suitable constructor for receiving baseString 204, sortString 206 and altString 208 from IME 224 via data entry methods 210 for entry into an IString object 202 may be:

```

*****
*
* <P></P>
*
* <dt><b>Description:</b><dd>
* <p> Allocate a new IString. The baseString, sortString and
* altString are entered from the IME utilizing the default language and
* locale.</p>
*
*****
public IString(String base,
                String sort,
                String alt,
                Locale src,
                Locale tgt) {
    this.baseString = base;
    this.sortString = sort;
    this.altString = alt;
    this.sourceLocale = src.getLocale();
    this.sourceLanguage = src.getLanguage();
    this.targetLocale = tgt.getLocale();
    this.targetLanguage = tgt.getLanguage();
    init();
}

```

The contents of baseString 204, sortString 206 and altString 208 are entered into the respective fields from data derived from IME 224, while the contents of sourceLocale 212 and sourceLanguage 214 are entered from the default locale and language properties specified by the host system in which data is being entered into IString object 202. The contents of targetLocale 216 and targetLanguage 218 will typically be a locale/language code for a language utilizing the latin character set such as "en_US" (English - United States).

Regardless of the language in which text is entered into an IString class object 202, the data automatically entered into each of the baseString 204, altString 206, and

sortString 208 by data entry methods 210 may be overridden or altered using other methods. The fields of an IString object 202 may preferably be individually and independently edited, allowing artificial promotion within sortString field 206 as described above, replacement of an erroneously selected ideograph in baseString field 204, or correction of a phonetic spelling within altString field 208.

While the above-described methods assumed that the source and target languages were taken from host system defaults, data may alternatively be entered into baseString 204, sortString 206 and altString 208 for specified source and target languages utilizing the constructor:

```

/*****
 *
 * <P> </P>
 *
 * <dt> <b>Description:</b> <dd>
 * <p> Allocate a new IString. The baseString, sortString and
 * altString are entered from the IME for specified target and source
 * language and locale. </p>
 *
 *****/
10 public IString(String base,
        String sort,
        String alt,
        String srcLanguage,
        String srcLocale,
        String tgtLanguage,
        String tgtLocale) {
15     this.baseString = base;
        this.sortString = sort;
        this.altString = alt;
        this.sourceLocale = srcLocale;
        this.sourceLanguage = srcLanguage;
        this.targetLocale = tgtLocale;
        this.targetLanguage = tgtLanguage;
        init();
25     }
}

```

In this constructor, the source and target language and locale which are employed to select the characters entered into baseString 204, sortString 206 and altString 208 may be specified. This latter constructor may be employed to create an IString object 202 in other than the host system default language, or in host systems where data for the IString object 202 is received from another system and a local instance is created.

It should be noted that transliteration engine 220 and messaging methods 226 need not necessarily be implemented within an IString class 202 as depicted in Figure 2, and

that IME method 224 need not be implemented separately. Transliteration engine 220 and messaging methods 226 may instead be implemented within separate subclasses which are appropriately constructed and/or invoked by IString class 202 as necessary, while IME 224 may be implemented as a method within IString class 202.

Transliteration engine 220 and IME 224 and are only required by data entry methods 210 to gather input data for IString class 202 objects under certain locale and language property settings. Otherwise, data may be programmatically input into baseString 204, sortString 206, and altString 208 by invoking the proper constructor. The methods which may be invoked by programs at runtime to programmatically get and set fields within IString 202 include:

```

/*****
 *
 * <P> </P>
 *
 * <dt> <b>Description: </b> <dd>
 * <p> Get the IString baseString. </p>
 *
 * @returns str String containing the base string
 *
 *****/
public String getBaseString() {
    return this.baseString;
}

```

This method returns the contents for baseString 204 for an IString object 202. Similar methods return the contents of sortString 206 and altString 208:

```

/*****
 *
 * <P> </P>
 *
 * <dt> <b>Description: </b> <dd>
 * <p> Get the IString sortString. </p>
 *
 * @returns str String containing the sort string
 *
 *****/
public String getSortString() {
    return this.sortString;
}

```

```

/*****
 *
 * <P> </P>
 *
 * <dt> <b>Description: </b> <dd>
 * <p> Get the IString altString. </p>
 *
 * @returns str String containing the alt string
 *
 *****/
public String getAltString() {
    return this.altString;
}

```

The methods also include setting baseString 204:

```

/*****
 *
 * <P> </P>
 *
 * <dt> <b>Description: </b> <dd>
 * <p> Set the IString baseString. </p>
 *
 * @param str String containing the base string
 *
 *****/
public void setBaseString(String sBase) {
    this.baseString = sBase;
}

```

as well as sortString 206 and altString 208:

```

5      /*****
      *
      * <P> </P>
      *
      * <dt> <b>Description: </b> <dd>
      * <p> Set the IString sortString. </p>
      *
      * @param str String containing the sort string
      *
10     *****/
      public void setSortString(String sSrt) {
          this.sortString = sSrt;
      }

15     /*****
      *
      * <P> </P>
      *
      * <dt> <b>Description: </b> <dd>
      * <p> Set the IString altString. </p>
      *
      * @param str String containing the alt string
      *
20     *****/
      public void setAltString(String sAlt) {
          this.altString = sAlt;
      }

25
30

```

In addition to getting and setting baseString 204, sortString 206, and altString 208 for an IString object 202, programs may need to get or set the display locale or language of an IString object 202. Accordingly, other methods are provided to permit a program to get and/or set the locale or language properties of IString data:

```

/*****
 *
 * <P> </P>
 *
5  * <dt> <b>Description:</b> <dd>
 * <p> Get the locale of the IString data.</p>
 *
 * @returns loc Locale containing the locale of the data
 *
10 *****/
public Locale getLocale() {
    Locale loc = new Locale(this.sourceLanguage, this.sourceLocale);
    return loc;
}

/*****
 *
 * <P> </P>
 *
 * <dt> <b>Description:</b> <dd>
 * <p> Set the locale of the IString data.</p>
 *
 * @param loc Locale of the data
 *
25 *****/
public void setLocale(Locale loc) {
    this.sourceLocale = loc.getLocale();
    this.sourceLanguage = loc.getLanguage();
}

```

```

/*****
 *
 * <P> </P>
 *
 * <dt> <b>Description:</b> <dd>
 * <p> Get the display language of the IString data.</p>
 *
 * @returns Display language of the data
 *
 *****/
public String getDisplayLanguage() {
    Locale loc = new Locale(this.sourceLanguage, this.sourceLocale);
    return loc.getDisplayLanguage();
}

/*****
 *
 * <P> </P>
 *
 * <dt> <b>Description:</b> <dd>
 * <p> Get the display locale of the IString data.</p>
 *
 * @returns Display locale of the data
 *
 *****/
public String getDisplayLocale() {
    if(this.sourceLanguage == null&&this.sourceLocale == null)
        return null;
    else{
        Locale loc = new Locale(this.sourceLanguage, this.sourceLocale);
        return loc.getDisplayLocale();
    }
}

```

While these methods are available, IString class 202 preferably exhibits a "black box" behavior such that the programmer/user need not know anything about the methods implemented for IString class 202. IString class 202 simply appears as a data type which encapsulates extra information about baseString 204 and also includes some methods for transforming characters from one character set to another.

For special cases where the sortString field 206 or altString field 208 are to be exposed to the user in addition to or in lieu of baseString 204, either for editing or for display only, a separate set of controls may be provided.

In the present invention, IString class 202 is employed to effectively transfer human language data across systems employing incongruous languages. The contents of baseString 204 provide a native representation of the text in the default language of the system originating the IString object 202. However, for each system participating in the exchange of data with other systems running in different human languages, the targetLocale property 216 and targetLanguage 218 property of an IString object 202 are preferably set to a common value (e.g., targetLocale="US", targetLanguage="en"). The contents of altString 208 will thus contain a common, cross-language representation of the text string. In systems where the default language of a system receiving an object differs from the language of the contents of baseString 204, IString class object 202 may automatically switch to presenting the contents of altString 208 as the text string to be displayed or processed.

Referring to **Figure 3**, a high level flowchart for a process of entering data into a multi-field text string class in accordance with a preferred embodiment of the present invention is depicted. **Figure 3** is intended to be read in conjunction with **Figure 2**. The process shown in **Figure 3** begins at step 302, which depicts initiation of data entry into a multi-field text string class (IString) object 202. The process then passes to step 304, which

illustrates a determination of whether the currently selected language, specified by the operating system or application environment language and/or locale properties, employs the latin alphabet character set. Languages supported by a given system may be categorized to facilitate this determination, and the category employing the latin alphabet should include English and the romance languages (Spanish, French, Italian, etc.). If the current language employs the latin alphabet character set, the process proceeds to step 306, which depicts inserting the text entered by the data entry keystrokes into all three fields--baseString 204, sortString 206, and altString 208--of IString object 202. Thus, in most locales and/or languages for a locale, data is input programmatically by invoking the appropriate constructor and the baseString text is inserted into the other two fields by default.

Referring back to step 304, if the currently-selected language for data entry into the IString object does not utilize the latin alphabet character set, the process proceeds instead to step 308, which illustrates a determination of whether the currently selected language maps to the latin alphabet character set. This category of languages will most likely include, for example, Cyrillic, Greek, Hebrew, and many Germanic and Arabic languages. If so, the process proceeds to step 310, which depicts storing the text of the data entry keystrokes into the baseString and sortString fields 204 and 206, and then to step 212, which illustrates storing the entered text sound-mapped to latin alphabet characters in the altString field 208.

5 It should be noted that there may be some overlap
between the first and second categories of languages, or--
stated differently--some language-dependent variation in the
manner in which language entry is handled. For example, for
Spanish text, while most characters may be entered directly
into all three fields of an IString class object, the "ñ"
character may be sound mapped to "ny" in the altString field
of an IString object to provide information regarding proper
pronunciation. Alternatively, the altString field may be
10 filled with a traditional phonetic pronunciation guide to
the data entered into the IString object (e.g., "küm-er" or
"koo-mer") to provide pronunciation information for words in
languages employing the latin alphabet character set as well
as for ideographs.

15 Referring back to step 308, if the current language
does not map readily to the latin alphabet character set
(e.g., the language employs an ideographic character set),
the process proceeds instead to step 314, which depicts
storing the data entry keystrokes in the altString field 208
as a pronunciation guide, then to step 316, which
illustrates storing intermediate characters (such as
hiragana or katakana characters) in the sortString field
206, and finally to step 318, which depicts storing the
ideograph in the baseString field 204. Steps 314, 316 and
25 318 illustrates the operation of the IME in storing data in
an IString object.

30 From any of steps 306, 312, or 318, the process
proceeds to step 320, which illustrates the process becoming
idle until data entry to an IString class object is again
initiated. The distinct data entry behavior of the IString

class based on the locale or language property allows language specific characters to be automatically mapped to recognizable characters and saved as a pronunciation key. A user may thus view the character-mapped representation of an abstract object name to be able to recognize a specific object, despite a lack of familiarity with the character set in which the object name text string was entered.

Referring to **Figures 4A** and **4B**, portions of a user interface showing one application for character-mapped data entry into alternate fields of a multi-field text string class in accordance with a preferred embodiment of the present invention are illustrated. The user interface display an IString object in which entered text was saved in the baseString field and transliterated characters for the entered text was automatically saved in the altString 208 field as described above.

In **Figure 4A**, the baseString field contents ("Давид Кумгир") is displayed. A user unfamiliar with the Cyrillic character set would not be able to recognize this name. However, by altering the user interface so that the altString field contents ("David Kumhyr") is displayed as shown in **Figure 4B**, the user can recognize the name of an object which they wish to manipulate. The automatically transliterated characters saved in the altString field provide a recognizable representation of the text string, as well as a pronunciation key.

With reference now to **Figure 5** and **Figures 6A** through **6G**, a high level flowchart and corresponding user interface displays for data entry in a logosyllabic language into a

multi-field text string class in accordance with a preferred embodiment of the present invention are depicted. **Figures 5 and 6A-6G** are intended to be read in conjunction with each other and with **Figure 2**, and depict the process of IME 224 in generating the contents of baseString 204, sortString 206, and altString 208 for an ideographic (or logosyllabic) language as described in connection with steps 314, 316, and 318 of **Figure 3**.

For Japanese and similar logosyllabic languages, IME 224 monitors the keystrokes entered, selecting appropriate hiragana (or katakana) characters, and finally presents a list of possible matching Kanji symbols. The process begins at step 502, which illustrates data entry being initiated with the language selected not mapping to the latin alphabet character set. The process may thus be performed between steps 308 and 314 depicted in **Figure 3**. From step 502, the process first passes to step 504, which illustrates a determination of whether a character has been entered by the user. Any character entry should be in latin alphabet characters, even for logosyllabic languages. As noted earlier, data input for such languages is accomplished phonetically utilizing combinations of latin alphabet characters to select symbols from phonetic syllabaries, with a dictionary lookup for the final ideograph.

If no character was entered, the process proceeds from step 504 back to step 504, to continue polling for character entry. If a character is entered, however, the process proceeds instead to step 506, which depicts adding the entered character into altString 208. The process then passes to step 508, which illustrates a determination of

whether the entered character, together with any previously entered characters which have not yet been mapped to a syllabary symbol, corresponds to a syllabary symbol. If not, the process returns to step 504 to await further character input. If so, however, the process proceeds instead to step 510, which depicts adding the syllabary symbol corresponding to the entered character(s) to altString field 206.

From step 510, the process then passes to step 512, which illustrates a determination of whether the entered character, together with any previously entered characters which have not yet been mapped to an ideograph, corresponds to an ideograph. If not, the process returns to step 504 to await further character input. If so, however, the process proceeds instead to step 514, which depicts presenting the (potentially) matching ideograph or ideographs to the user for selection, then adding the selected ideograph to baseString 204. The process then returns to step 504 to await further input, and continues as described until interrupted by a control indicating the data entry is completed or terminated. Those skilled in the art will recognize that some mechanism should be provided for prevented termination of data entry when latin alphabet characters entered to phonetically compose an ideograph do not correspond to any available ideograph in the dictionary.

A specific example of data entry utilizing IME 224 is illustrated for the data in Table III by Figures 6A through 6G. In composing the word "hayashi," the user would first enter "h". This is not a valid hiragana character, so IME 224 will display the "h" as shown in Figure 6A, add the "h"

to the altString field 208, and wait for the next character to be entered. The user next enters "a", forming the phoneme "ha", for which IME 224 may temporarily display "ha", then select and display "は" in lieu of "ha". Alternatively, IME 224 may simply select and display "は" in lieu of "h" as shown in **Figure 6B**. IME 224 also adds the "a" to the altString field 208 and adds "は" to sortString field 206.

Similarly, upon entry of "y" by the user, IME 224 displays the "y" as shown in **Figure 6C** and adds that character to altString 208; upon entry of "a", IME 224 selects and displays "や" in place of the "y" as shown in **Figure 6D**, then adds "a" to altString 208 and "や" to sortString 206. Following user entry of "s", IME 224 adds the character to altString 208 and displays the character as shown in **Figures 6E**; when the user subsequently enters "h", IME 224 adds the character to altString 208 and displays the character as shown in **Figures 6F**.

Finally, upon user entry of "i", which causes the entered text to correspond to a phonetic spelling of a Kanji character, IME 224 adds "i" to altString 208 and selects and adds "い" to sortString 206. As each latin character is entered by the user and added to altString 208, or alternatively as each hiragana/katakana syllabary symbol is selected and added to sortString 206, IME 224 checks the lookup dictionary for possible Kanji symbols corresponding to the phonetic pronunciation entered. Upon determining a match, IME 224 presents a list to the user for selection as shown in **Figure 6G**. The user selection is subsequently entered into baseString 204.

Filtering input to IString object 202 through IME 224, which may form a portion of the operating system, the application environment, or an editor, allows intermediate and alternative representations of a name or word to be captured along with the final form. This is particularly important for logosyllabic languages using ideographs, which are especially difficult to deal with for non-speakers of that language. Describing an ideograph over the telephone for assistance in determining meaning can be extremely challenging, a task further complicated where the same ideograph has multiple meanings or pronunciations. For instance, many Kanji symbols have multiple pronunciations and/or meanings. Therefore, merely seeing the characters does not provide enough information to know how to pronounce the name, which has resulted in the Japanese business card ritual of presented the card and pronouncing the name at the same time. Capturing intermediate representations within IString object 202 allows non-speakers to read and know how to pronounce or match ideographic characters.

It is important to note that while the present invention has been described in the context of a fully functional data processing system and/or network, those skilled in the art will appreciate that the mechanism of the present invention is capable of being distributed in the form of a computer usable medium of instructions in a variety of forms, and that the present invention applies equally regardless of the particular type of signal bearing medium used to actually carry out the distribution. Examples of computer usable mediums include: nonvolatile, hard-coded type mediums such as read only memories (ROMs) or erasable, electrically programmable read only memories

(EEPROMs), recordable type mediums such as floppy disks, hard disk drives and CD-ROMs, and transmission type mediums such as digital and analog communication links.

- 5 While the invention has been particularly shown and described with reference to a preferred embodiment, it will be understood by those skilled in the art that various changes in form and detail may be made therein without departing from the spirit and scope of the invention.

005727- E067260